PATENT

DOCEKT NO.: INTEL 2207/16369

UNITED STATES PATENT APPLICATION
FOR

METHOD FOR PAGE SHARING IN A PROCESSOR WITH MULTIPLE THREADS AND PRE-VALIDATED CACHES

INVENTORS:

SAILESH KOTTAPALLI
NADEEM FIRASTA

PREPARED BY:

KENYON & KENYON
333 WEST SAN CARLOS STREET, SUITE 600
SAN JOSE, CALIFORNIA 95110

(408) 975-7500

Express Mail No.: EV351181194US
Docket No.: Intel 2207/16369

METHOD FOR PAGE SHARING IN A PROCESSOR WITH MULTIPLE THREADS AND PRE-VALIDATED CACHES

Background of the Invention

[0001] The present invention pertains to a method and apparatus for page sharing in a multithreaded processor. More particularly, the present invention pertains to the maintenance of a translation look-aside buffer with pre-validation of physical addresses.

[0002] As is known in the art, a translation look-aside buffer (TLB) is used to map a virtual memory address to a physical memory address. A programming thread, executed by a processor, initiates a read or update to a physical memory address by providing a virtual address. The processor searches the TLB for the virtual address, retrieves the physical address, and computes the physical address tag for a given cache. The physical address tag for the matching index is retrieved from the cache. A full comparison is executed between the tag of the request and the cache tag. If the cache tag matches the request tag, the search is registered as a hit. If the cache tag does not match the request tag, the search is registered as a miss. The full comparison of the tags and the full retrieval of the physical address from the TLB require a great deal of processor time.

[0003] Problems begin to occur when more than one thread is supported in the processor core. Two threads using the same virtual and physical page would require different TLB entries to support different page "access" rights. This defeats the ability to share common memory contents between threads since each thread requires the presence of its own translation entry in the TLB. Thus a request to a line from one thread is unable to get a cache-hit on a line cached by a different thread.

Brief Description of the Drawings

[0004] **Figure 1** is a block diagram of a portion of a processor employing an embodiment of the present invention.

[0005] **Figure 2** is a flowchart showing an embodiment of a method according to the present invention.

[0006] **Figure 3** provides an illustration of one embodiment of a processor system according to the present invention.

Detailed Description of the Drawings

[0007] A method and system for allowing a multi-threaded processor to use a translation look-aside buffer (TLB) and a cache with pre-validated tags are described herein. The multi-threaded processor may search a translation look-aside buffer in an attempt to match a virtual memory address. If no matching valid virtual memory address is found, the processor retrieves the appropriate translation from the next level TLB or virtual hash page table (VHPT). During insertion, the translation look-aside buffer may be searched for a matching physical memory address. If a matching physical memory address is found, the associated virtual memory address may be overwritten with a new virtual memory address. The multi-threaded processor may execute switch on event multi-threading or simultaneous multi-threading. If simultaneous multi-threading is executed, then access rights for each thread may be associated with the translation.

[0008] One method of speeding up the retrieval process is the use of 1-hot vectors. In general, a cache that stores 1-hot vectors as tags is referred to as a 1-hot cache tag. A 1-hot vector is an n-bit string that contains a single "1" and n-1 "0's", for example, "00001000" is an eight-bit 1-hot vector. The 1-hot vector has the same number of bits as the number of entries in the TLB. When a line is cached, instead of tagging the line with the conventional tag, the line is tagged with the 1-hot vector that points to the TLB entry that contains the translation for the physical address corresponding to the line. When a memory request, with a virtual address, searches the TLB, the content addressable memory (CAM) lookup of the virtual addresses of all translations stored in the TLB returns a 1-hot match vector pointing to the entry that contains the translation for the request. The 1-hot vector corresponding to the index portion of the memory request is retrieved from the 1-hot cache tag. The 1-hot vector generated from the TLB match for the request and the 1-hot vector corresponding to the request index of the 1-hot cache tag is

compared using an AND-OR operation on each bit. A "true" result indicates a cache hit. A "false" result indicates a cache miss.

[0009] The 1-hot scheme requires that no two different virtual addresses, or "synonyms", be mapped to the same physical address in the TLB. This is because the cache tag is generated using the physical address CAM on the TLB during cache fill while the tag for the memory request is generated on the CAM match on the virtual address of the TLB. Therefore, each physical address mapping in the TLB must be unique. To support this, a "column clear" needs to be executed on the 1-hot vector in the cache tag each time a new entry is added (and the old one replaced) in the TLB. The column clear removes the possibility of cache hit for lines mapping to the physical address translation provided by the old (replaced) translation.

[0010] **Figure 1** illustrates in a block diagram one embodiment of a processor 100 using the TLB 110. The processor may execute a memory instruction using a memory execution engine 120. The processor may be a multithreaded processor, capable of running multiple programming threads. The multiple programming threads may be executed using either simultaneous multi-threading (SMT) or switch on event multi threading (SoEMT). In SMT, the multiple programming threads may be executed concurrently. In SoEMT, the multiple programming threads may be executed in an alternating fashion. A first thread is executed until an event occurs, such as a long latency stall, after which the processor moves the active thread to the background and switches to an inactive thread.

[0011] The TLB 110 may be configured to store a number of translations, each translation containing a virtual memory address 111 and physical memory address 112. If the multi-threaded processor 100 uses SMT, a first set of access rights 113 may be stored in the TLB 110 and associated with the translation. The first set of access rights 113 may refer to the ability

of a first programming thread to execute an operation using the data stored at that physical memory address 112. The operations may include "read", "write", "execute", or other processing operations. A second set of access rights 114 may be stored in the TLB 110 and associated with the same translation. The second set of access rights 114 may refer to the ability of a second programming thread to execute an operation using the data stored at that physical memory address 112. If the second programming thread uses a different translation for the physical memory address 112, the first set of access rights 114 are erased when the second programming thread overwrites that translation. If the multi-threaded processor 100 uses SoEMT, access rights for each thread may be stored in the TLB 110. However, as the threads in SoEMT have control of the TLB 110 for greater periods of time this is not as necessary.

[0012] The processor 100 may use the TLB 110 to access data in a data cache 130. The TLB 110 translates a virtual memory address 111 into a physical memory address 112 of the memory cache 130. The processor may use a first content address memory (CAM1) 140 to search the virtual memory addresses stored in the TLB 110. If the first CAM 140 is unable to find the virtual memory address 111, a miss may be returned and a new translation for the physical address 112 may be inserted into the TLB 110. For the insertion, the processor may use a second content address memory (CAM2) 145 to search the physical memory addresses 112 stored in the TLB 110.

[0013] The data cache may be divided into a number of ways. Data may be stored in any way of the cache-set matching the index of the address. Each address identifies a set as well as a byte within each cache line. The lower bits in both the virtual memory address 111 and the physical memory address 112 may indicate at which set and byte within a line the data is located. The remaining upper bits in the physical memory address may be the tag for that address. When

a memory instruction is executed, a first de-multiplexer 150 may select the data at that particular set for each way. The data cache 130 may be associated with a cache tag 160. A second de-multiplexer 170 may select the tag in the cache tag 160 at that address for each way. A comparator 180 compares each tag from the cache tag 160 with the physical memory address 112 retrieved from the TLB 110 to determine which way, if any, has the matching tag. The comparator 180 signals a multiplexer 190, which receives the data at that address for each way, and sends on the data with the matching tag.

[0014] Alternatively, the cache tag 160 may contain a 1-hot vector associated with the position of the translation in the TLB 110 computed when the translation is inserted into the TLB 110. Upon retrieval of the data in the data cache 130, the 1-hot vector from the cache tag 160 is compared to the position of the translation in the TLB 110 to determine if the retrieved data is valid. The 1-hot vector allows the comparator 180 to be a simple AND-OR comparison of the stored 1-hot vectors and the position of the translation within the TLB 110. Additionally, if a new translation is inserted into the TLB 110 by overwriting a translation with the same physical address 112, the column clear of the cache tag 160 does not have to be performed as the 1-hot vector should still be valid.

[0015] **Figure 2** illustrates in a flowchart one embodiment of a method of replacing physical addresses within the TLB 110. The process may start (Block 205) by executing a first memory thread with a memory execution engine 120 (Block 210). The memory execution engine 120 may search the virtual memory addresses 111 of the TLB 110 for a virtual memory address that matches the virtual memory address of the thread using CAM1 140 (Block 215). If a valid virtual memory address is found (Block 220), the memory cache 130 at the physical memory address may be accessed and the next instruction of the first programming thread may

be executed (Block 210). If no valid virtual memory address is found (Block 220), the proper translation may be retrieved from a second level TLB or VHPT (Block 225). The memory execution engine 120 may search the physical memory addresses 112 of the TLB 110 for a matching physical memory address using (Block 230). If a matching physical memory address 112 is found, the associated translation is overwritten by a new translation (Block 235). If no matching physical memory address 112 is found, a new slot within the TLB 110 is found in which to insert the new translation (Block 240). A column clear is executed upon the 1-hot vector to clear the 1-hot vector in the cache-tag indexing to the replaced translation (Block 245). The new translation is then inserted into the proper slot (Block 250). The memory execution engine 120 then executes the next instruction in the first memory thread (Block 210), or the second memory thread if a switch is called for.

[0016] **Figure 3** shows a computer system 300 that may incorporate embodiments of the present invention. The system 300 may include, among other components, a processor 310, a memory 330 and a bus 320 coupling the processor 310 to memory 330. In this embodiment, processor 310 operates similarly to the processor 100 of Fig. 1 and executes instructions provided by memory 330 via bus 320.

[0017] Although a single embodiment is specifically illustrated and described herein, it will be appreciated that modifications and variations of the present invention are covered by the above teachings and within the purview of the appended claims without departing from the spirit and intended scope of the invention.